AIBridge

Lecture 1

AIBridge

- Bridge the gap between AI and [your choice]
- First camp at UC Davis in June 2022, 2nd in Silicon Valley in March 2023
- Acquire basics: Python, basic ML algorithms, toolbox usage
- Enable further learning
- Enable easier communications and collaborations

- AIFS - NSF/USDA AI Institute for Next Generation Food Systems

AI in Food Systems

- Molecular breeding
  - Help breeders to run more efficient and targeted breeding programs
- Agricultural production
  - Crop yield sensing and forecasting
  - Water and nitrogen stress sensing, prediction, accusation
- Food processing
  - Tomato processing loss prediction
  - Sanitation classification
- Nutrition
  - Use food photo and text to predict core ingredients
  - Dietary recommendation

WHAT IS AI/ML?

AI vs. ML

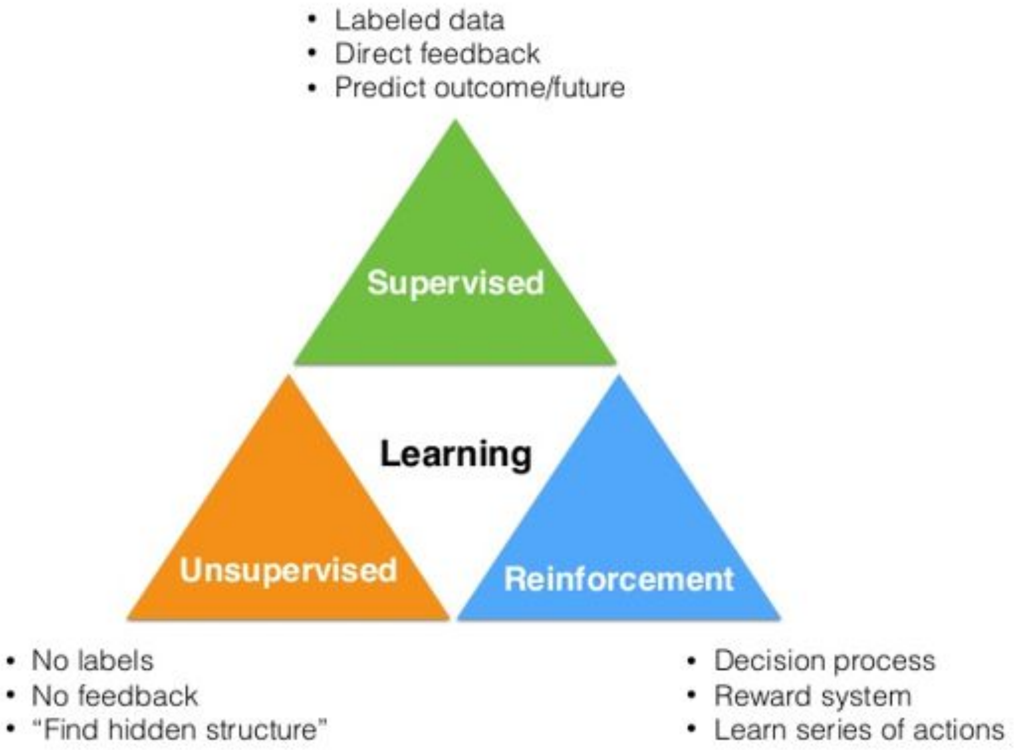
What can AI do

Data → **AI** → **Info**

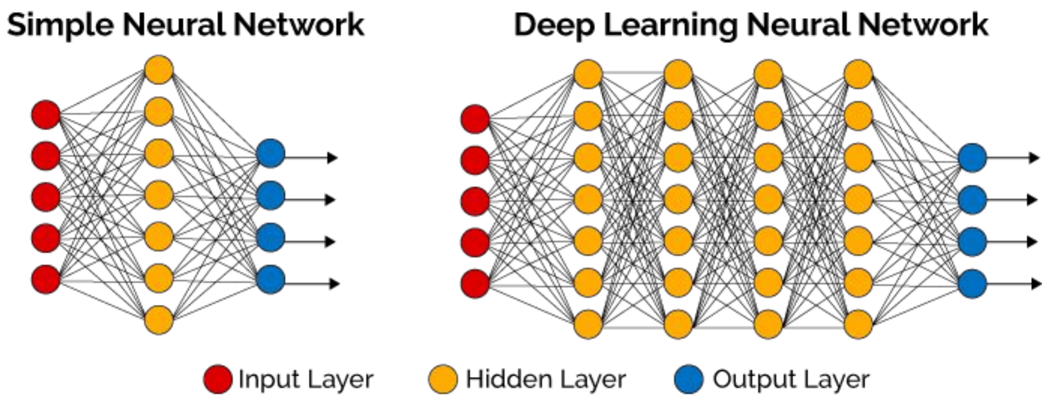
Machine Learning

- Arthur Samuel (1959). Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed.
- Tom Mitchell (1998) Well-posed Learning Problem: A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

A High-Level View

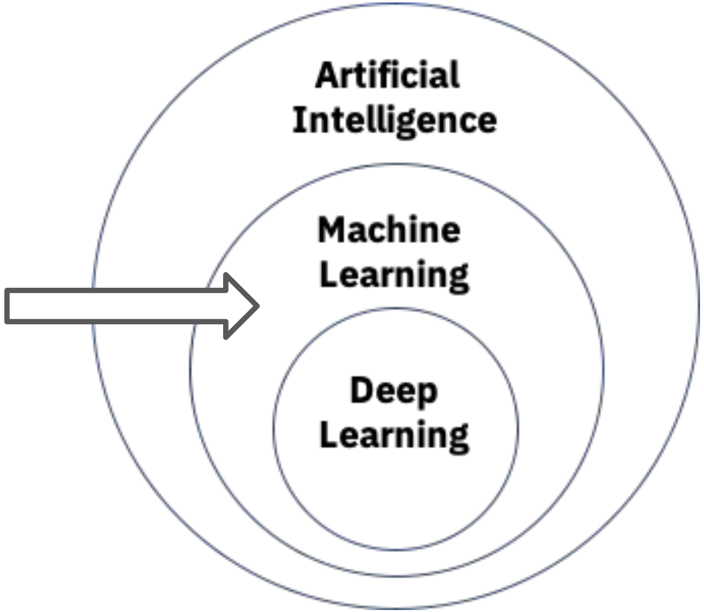


Deep Learning




CHATGPT
OpenAI

Our focus



Class Structure

- Lecture + break + lab
  - Lab is the best part of this bootcamp
- Recap
  - Overview of key knowledge points
  - Feedback from you (pace, clarity, etc.)
- Learning by doing
  - Iris dataset
  - Wine dataset
- Go through the process to complete a basic ML project

Schedule

- Python: 1.5 days
  - Condensed with a focus on what we need for ML
- ML: 3 days
  - More intuitions
- Friday afternoon: Shark Tank

Schedule

	Monday	Tuesday	Wednesday	Thursday	Friday
9:00 - 10:20	**Lecture 1** Python Basic Syntax	**Lecture 3** Functions and Documentation	**Lecture 5** Accuracy Precision Recall and Data	**Lecture 7** Overfitting and Feature Selection	**Lecture 9** ChatGPT for Coding and AI
10:20 - 10:30	Break	Break	Break	Break	Break
10:30 - 12:00	Lab 1	Lab 3	Lab 5	Lab 7	Presentation Prep
12:00 - 1:00	*Lunch Break*	*Lunch Break*	*Lunch Break*	*Lunch Break*	*Lunch Break*
1:00 - 2:20	**Lecture 2** List Manipulation, OOP, and IO	**Lecture 4** Intro to Regression and Classification	**Lecture 6** Three Additional Classifiers	**Lecture 8** Unsupervised Learning Algorithms	Presentation Prep
2:20 - 2:30	Break	Break	Break	Break	Break
2:30 - 4:00	Lab 2	Lab 4	Lab 6	Lab 8	**Presentations**

Typical Practices in ML/Programming

- Find a sample
- Read through it
- Try it
- Modify it
- Google it

- Basic skills to do these and practice them

Best Practices

- Ask questions
- Type along during lectures
- Ask for help
- Make good use of labs
- Provide feedback

Learning by Doing

- Iris
- Wine
- Your own on Day 5 PM

Resources

- Class notes, links in notes
- Python: https://www.w3schools.com/python/
- Sklearn user guide: https://scikit-learn.org/stable/user_guide.html
- Google
- ChatGPT*

INTRODUCTION TO PYTHON

Python

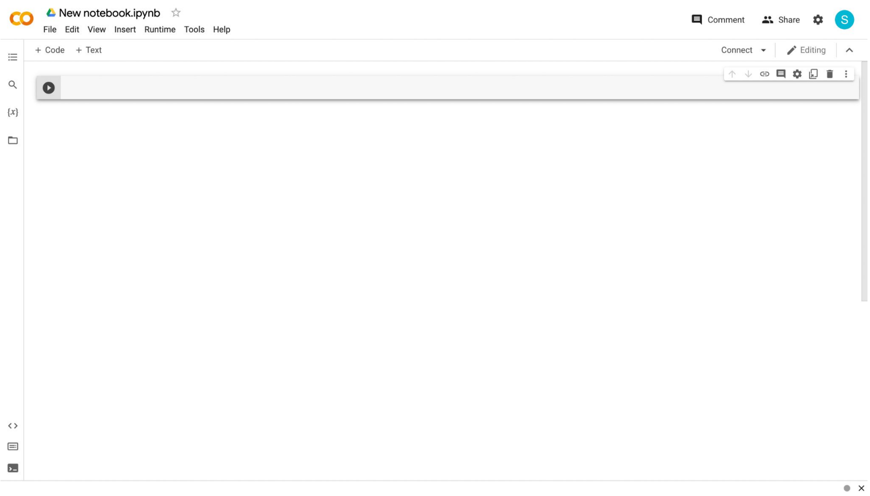
- Python is a popular programing language
- Guido van Rossum, Dutch programmer, invented in late 1980s
- Widely used in industry and academia, especially for ML applications.
- R vs Python
  - Python better at large data amounts and machine learning

Lecture Outline

- Google Colab
- General Python Syntax
- Variables
- Logic
- Control Flows

Google Colab Setup

- https://colab.research.google.com/
- Stores everything on Google Drive
- Can be shared with others and across devices
- No setup required
- Most packages/libraries preinstalled

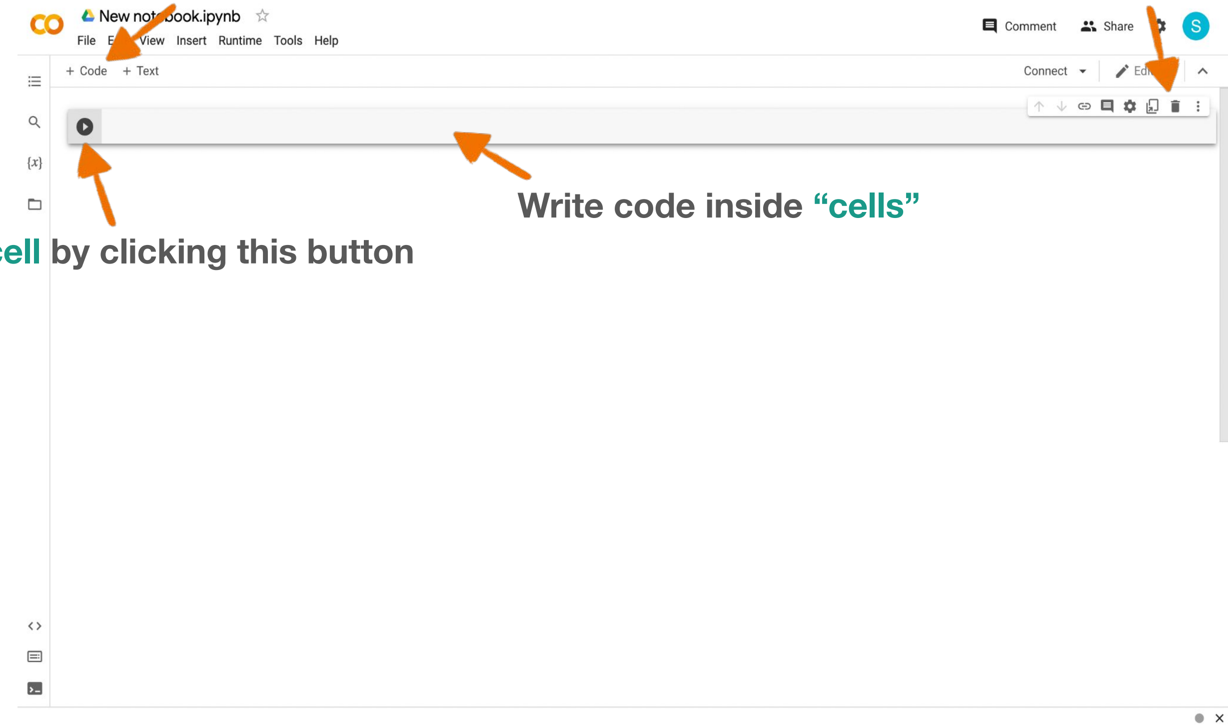


Follow along as we work through the Python language

Google Colab UI

Add new **cells** by clicking "+ Code"

Delete **cells** by clicking this button



Write code inside **"cells"**

Run a **cell** by clicking this button

Lecture Outline

- Google Colab
- General Python Syntax
- Variables
- Logic
- Control Flows

Getting Started

- Comments allow sections of the code to be more readable
  - Anything after a “#” is a comment
  - `# I am a comment!`

- Functions take in inputs and give outputs
  - `print(`*`input`*`)`
    - The print function prints out the input
  - `print("hello world")`

Lecture Outline

- Google Colab
- General Python Syntax
- Variables
- Logic
- Control Flows

Overview

- A variable is a reserved place in memory (think: container) which can store a **value**
    - Creating variables: `variable_name = value`
- Can be used anywhere after its assignment, but never before
- Can re-assign values as needed
- 7 types of values: Integer, Floating-point, String, Boolean, List, Tuple, and Dictionary
    - (More details about each type coming up in next slides)

```
var_a = 25

var_a = 70

print(var_a)
```

Names

- Cannot start with a number
- Cannot include spaces
- Cannot be a keyword: https://www.w3schools.com/python/python_ref_keywords.asp
- Should be descriptive
- *Good practice: all lowercase with underscores for spacing

"~~3rd_variable~~"

"~~my variable~~"

Good examples: `datapoint_number, petal_width, ...`

Self-Test

What does the following code output?

```
variable_a = 25

varaible_b = 70

variable_a = 40

variable_b = variable_a

print(variable_b)
```

A. **70 ⇒ because the value of** `variable_b` **is set to be 70 in the second line**

B. **40 ⇒ because the value of** `variable_b` **is set to be the same as** `variable_a` **which is 40**

C. **25 ⇒ because the value of** `variable_b` **is set to be the same as** `variable_a` **which is 25**

Self-Test

What does the following code output?

```
variable_a = 25

varaible_b = 70

variable_a = 40

variable_b = variable_a

print(variable_b)
```

A. 70 ⇒ because the value of `variable_b` is set to be 70 in the second line

B. 40 ⇒ because the value of `variable_b` is set to be the same as `variable_a` which is 40

C. 25 ⇒ because the value of `variable_b` is set to be the same as `variable_a` which is 25

Integer

- Non-fractional number
- Positive or negative
- No maximum or minimum practically

```
first_number = 1
second_number = 5
third_number = -3
```

Floating-Point

- “Float”
- Decimal point number
- Accurate within 2^{-55}

```
petal_length = 3.5

petal_width = 4.0
pi = 3.14159265358
```

3.1415926

Floating (Decimal) Point

String

Not this

- A string of characters
- **Put in quotations `"  "` or `'  '`**
- *Block string (multi-line string): three quotation marks
- *Special character (new line): `"\n"`

```
first_string = "s"
second_string = "string 2"
second_string = "another string"
```

Boolean

- True or False (capitalize)

```
first_boolean = True
second_boolean = False
```

Variables

List

- A list of values
  - `my_list = [value_1, value_2, ...]`
  - `example_list = [5, 20, 11, 3, 10]`
  - Can include multiple different data types
    - `multi_type_list = ["hello world", True, 5]`
- For a specific value in the list: `my_list[index]`
  - The index of the 1st item is 0,
  - `a_value = my_second_list[2] # gets the THIRD value in the list`
  - *There is also negative indexing (index of -1 gets last element, -2 gets second from last, etc.)

[a, b, c, d, e]

0 1 2 3 4

Self-Test

What does the following code output?

```
my_list = [21, 22, 23, 24, 25]

value = my_list[2]

print(value)
```

A. 22 ⇒ because value is set to the second item in the list

B. 23 ⇒ because value is set to the third item in the list

Self-Test

What does the following code output?

```
my_list = [21, 22, 23, 24, 25]

value = my_list[2]

print(value)
```

A. 22 ⇒ because value is set to the second item in the list

B. 23 ⇒ because value is set to the third item in the list

* Tuple

- Works the same as a list, but can't be changed
- Can contain multiple different data types

```
my_first_tuple = (object_1, object_2, ...)

my_second_tuple = (22, "hello!", True, 3.1415)

a_value = my_second_tuple[2] # gets the THIRD value in the tuple
```

* Dictionary

- A list of values with custom keys that are indices, like a list but indices are keys and not positions

```
my_dictionary={'apple':'fruit', 'banana':'fruit', 'cabbage':'vegetable',
'dragonfruit':'fruit','eggplant':'vegetable'}

print(my_dictionary['cabbage'])
```

Type Conversion

- Types: int, float, str, bool, list, tuple
- Convert types of variables to other types

```
my_float = float(my_string) #gives string in float form if possible
```

- Compatible types:
  - int → float
  - float → int (always rounds down)
  - str → int
  - str → float
  - *[most types] → string
  - *list → tuple
  - *boolean → int/float (0 → False, anything else → True)
  - *str → list/tuple (only converts str to list/tuple of single characters)

Variables

Basic Arithmetic Operations

+	-	*	**	/	//	%
Addition	Subtraction	Multiplication	Exponentiation	Division (turns int to float)	Floor Division (rounds down the quotient)	Modulus (returns the remainder)
`x + y`	`x - y`	`x * y`	`x ** y`	`x / y`	`x // y`	`x % y`
`1 + 2 == 3`	`2 - 1 == 1`	`2 * 3 == 6`	`2 ** 3 == 8`	`8 / 2 == 4.0`	`9 // 4 == 2`	`10 % 4 == 2`

Note: the double equal sign `a == b` is used to check for equality instead of assigning variables

Basic Arithmetic Operations

Changing a variable’s value:

```
x = 4
x = x + 1
# x becomes 5
```

```
x = 4
x = x - 2
# x becomes 2
```

```
x = 4
x = x * 2
# x becomes 8
```

Lecture Outline

- Google Colab
- General Python Syntax
- Variables
- Logic
- Control Flows

Conditionals

```
if statement_1:
  Code segment 1
elif statement_2: # elif means else if
  Code segment 2
else:
  Code segment 3
```

Example code

```
x = 1
y = 1

if x == y:
  print('x is equal to y')
elif x > y:
  print('x is greater than y')
else:
  print('x is less than y')
```

Example code

```
x = 4
y = 1

if x == y:
  print('x is equal to y')
elif x > y:
  print('x is greater than y')
else:
  print('x is less than y')
```

Example code

```
x = 4
y = 10

if x == y:
  print('x is equal to y')
elif x > y:
  print('x is greater than y')
else:
  print('x is less than y')
```

Logic Operations

`==` `!=` `<` `>` `<=` `>=`

`==` True if the two sides are exactly the same (`1 == 1` is True)

`!=` True if the two sides are NOT the same (`2 != 1` is True)

Logic Operations

- and: only runs if both are True

```
if 1 == 1 and 1 == 2:
    code segment…
```

- or: runs if at least one of them are True

```
if 1 == 1 or 1 == 2:
    code segment…
```

```
x = 4
y = 4
if x < y or x == y:
  print("x is less than or equal to y")
```

Self-Test

Which of these conditions are successfully passed?

```
petal_width = 1.8
petal_length = 3.5

if petal_width < 3 or petal_length < 3:
 print("condition 1 passed")

if petal_width < 3 and petal_length < 3:
 print("condition 2 passed")

if petal_width < 3:
 if petal_length < 3:
   print("condition 3 passed")
```

Self-Test

Which of these conditions are successfully passed?

```
petal_width = 1.8
petal_length = 3.5

if petal_width < 3 or petal_length < 3:
 print("condition 1 passed")

if petal_width < 3 and petal_length < 3:
 print("condition 2 passed")

if petal_width < 3:
 if petal_length < 3:
   print("condition 3 passed")
```

Lecture Outline

- Google Colab
- General Python Syntax
- Variables
- Logic
- Control Flows

Hypothetical Scenario

We have this very large list of 11 words:

```
word_list = ["Lorem", "ipsum", "dolor", "sit", "amet", "fusce",
"rhoncus", "mi", "viverra", "velit", "mattis"]
```

How do we access and print out every word?

Hypothetical Scenario

```
word_list = ["Lorem", "ipsum", "dolor", "sit", "amet", "fusce", "rhoncus", "mi",
"viverra", "velit", "mattis"]
```

```
print(word_list[0])
print(word_list[1])
print(word_list[2])
print(word_list[3])
print(word_list[4])
print(word_list[5])
print(word_list[6])
print(word_list[7])
print(word_list[8])
print(word_list[9])
print(word_list[10])
```

Horribly inefficient

A lot of tedious manual coding

Completely unscalable (what if there were 70 words)

Hypothetical Scenario

```
print(word_list[0])
print(word_list[1])
print(word_list[2])
print(word_list[3])
print(word_list[4])
print(word_list[5])
print(word_list[6])
print(word_list[7])
print(word_list[8])
print(word_list[9])
print(word_list[10])
```

Only difference between all these lines is the index

For Loops

- How to use: `for iterator in iterable:`
  - String, list, range, etc.
  - Need indentation

```
word_list = ["Lorem", "ipsum", "dolor", "sit", "amet", "fusce",
"rhoncus", "mi", "viverra", "velit", "mattis"]

for number in range(0, 11): #range goes through 0, 1, 2, … 10
  #this loop repeats 11 times and number changes to each number
  print(word_list[number])
```

For Loops

```
word_list = ["Lorem", "ipsum", "dolor", "sit", "amet", "fusce",
"rhoncus", "mi", "viverra", "velit", "mattis"]

for number in range(0, 11): #range goes through 0, 1, 2, … 10
  #this loop repeats 11 times and number changes to each number
  print(word_list[number])

for word in word_list:
  #this loop does the exact same thing but with less typing
  print(word)
```

For Loops

Output:

Lorem

```
word_list = ["Lorem", "ipsum", "dolor", "sit", "amet",
"fusce", "rhoncus", "mi", "viverra", "velit", "mattis"]

for word in word_list:
  #this loop does the exact same thing but with less typing
  print(word)
```

Self-Test

```
big_list = ["Lorem", "Ipsum", "Dolor", "Sit", "Amet",
"Consectetur", "Adipiscing", "Elit", "Sed"]
```

Which of the following code blocks will print out everything in the list?

a.

```
for word in big_list:
 print(word)
```

b.

```
for i in range(9):
 print(big_list[i])
```

c.

```
for word in big_list:
 print(big_list[word])
```

Self-Test

```
big_list = ["Lorem", "Ipsum", "Dolor", "Sit", "Amet",
"Consectetur", "Adipiscing", "Elit", "Sed"]
```

Which of the following code blocks will print out everything in the list?

```
a.

for word in big_list:
 print(word)
```

```
b.

for i in range(9):
 print(big_list[i])
```

```
c.

for word in big_list:
 print(big_list[word])
```

While

- How to use: `while statement:`
  - The loop repeats as `statement` is true
  - Needs indentation

```
my_number = 0
while my_number < 6:
  print(my_number)
  my_number = my_number + 1
```

Indentation

Don't worry about what this code does.

```
a_list = [3, 22, 1, 73, 40, 3, 19]
sum = 0

for i in range(0, 7):
    sum = sum + a_list[i]
    sum = sum / 2.4
    sum = sum * -1

    print(a_list[i])

print(sum)
```

Inside loop because of indentation *(tab)*